INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992 25ème anniversaire

N° 1761

Programme 3
Intelligence artificielle, Systèmes cognitifs et Interaction homme-machine

FONT METRICS

Jacques ANDRÉ

Octobre 1992

RR_1761

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

Font Metrics

Les Métriques de Fontes

Publication Interne n°676 - Septembre 1992 - 20 pages

Jacques ANDRÉ
Irisa/Inria-Rennes
Campus de Beaulieu, F-35042 Rennes cedex
jandre@irisa.fr

Projet OPERA/DIDOT
Programme 3

Abstract: This note introduces the basics of font metrics. Units used in typography are presented. Traditional notions such as *body size* are introduced. A rapid overview over formatting and printing functions shows where metric information is needed. Metrics related to horizontal and vertical placement of characters are described in detail, as well as problems related to kerning pairs and composite characters. A detailed explanation of AFM metrics files is provided.

Résumé On présente les notions de base de la métrique des fontes. Après avoir rappelé les diverses unités employées dans ce domaine et les notions de base (comme celle de *corps*), on montre pourquoi cette notion de métrique est indispensable aujourd'hui. On insiste en particulier sur les problèmes d'alignement vertical ou horizontal mais aussi sur les techniques de crénage ou de composition de caractères. Le format AFM d'Adobe sert de modèle pour les exemples.

Font Metrics*

Jacques ANDRÉ

Abstract

This note introduces the basics of font metrics. Units used in typography are presented. Traditional notions such as *body size* are introduced. A rapid overview over formatting and printing functions shows where metric information is needed. Metrics related to horizontal and vertical placement of characters are described in detail, as well as problems related to kerning pairs and composite characters. A detailed explanation of AFM metrics files is provided.

While many books are dedicated to typography or fonts, only a few discuss font metrics. The following books and papers provide some contributions related to type metrics [Karow87], [Rubinstein88], [André91], [Adobe91] chapter 5, [Black90]. Large type catalogues may also give an introduction to font metrics [Holthusen90].

1 Units

Measurements are expressed both in type size (what is to be measured) and in typographic point size (in which unit). Alas, the confusion surrounding these subjects has not been reduced by the advent of computer technology.

A summary of the history of typographic point definitions [Tracy86] helps to understand the problem.

*This paper will appear as a chapter in Roger Hersch (ed.), *Visual and Technical Aspects of Types*, Cambridge University Press, 1992. It was given as a lecture during the "First European Summer School in Digital Typography", Lausanne, 22–28 September, 1991. This research was partly funded by the CEE/Comett/Didot project number 90/3697Cb.

- The first attempt to define a rational way of measuring type area and line lengths has been made by 1723, in France. In 1725, the *point Fournier* was defined.

- F.- A. Didot defined his famous point in about 1775 as "one sixth of the *ligne de pied du Roy*" which is 1/72 of the french inch[1]. Since then, the *point didot* has been widely used in Continental Europe.

- In Britain[2] and in the United States, type sizes were given names[3] instead of values, at least up to 1886 when the American *Type Founder Association* defined the typographic point as 1/72 of an inch and the related Pica as 12 points.

- When the valve of the inch was reassessed, the typographic point kept its former value, so its definition became 1/72.27 of the inch.

- Finally, since 1980, most computations have been made by using a point rounded to 1/72 of the inch. As in [Black90], let us call this point the *DTP point (DeskTop Publishing point).*

Today, three main different systems exist (table 1).

The ratio between these values is not very large, however it may produce significant differences from one system to another (figure 1). Indeed, the Pica point is 7% smaller than the didot point: whereas 50 pica characters can fit in a line, only about 47 didot point characters could do so. Today's Desktop Publishing Systems generally follow PostScript conventions which use the DTP point (i.e. 1/72 of inch). Most professional formatters are able to compute sizes in any of these three systems.

Other systems are in use (e.g. the German standard DIN 16507 that uses millimeters) and new systems are frequently proposed.

---

[1]The metric system was only adopted in 1801 and a "millimetric point" was then defined, in 1811, as 0,40 mm: it seems that only the French *Imprimerie nationale* has ever used it.

[2]This was true as well in France and some names were still in used far after the didot metric, e.g. *dimant* stands for 3 points, *nonpareille* stands for 6 points and *petit parangon* stands for 18 points

[3]Such as "brillant" for 4 points, "nonpareil" for 6 points, "pica" for 12 points and "columbian" for 16 points. The same method was used for paper up to the standard definitions such as A4. The same is still in pratice today, even with computer fonts: the boldness of a character is defined by names such as "bold", "extrabold", "light", etc. although this could change thanks to new fonts such as Adobe's Masters.

1 didot point		0,3759 mm
1 cicero = 1 douze	= 12 didot points	4,5109 mm
1 Pica point	= 1/72.27 inch	0,3515 mm
1 Pica	= 12 Pica points	4,2175 mm
1 DTP point	= 1/72 inch	0,3528 mm

Table 1: The three major typographical points (1 inch = 25,4000 mm)

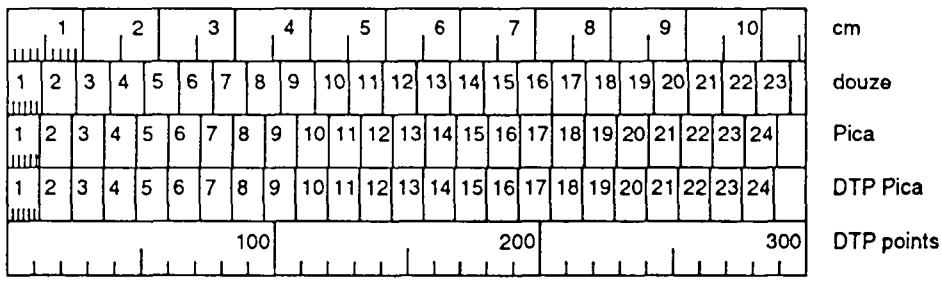

Figure 1: Comparison of the three point systems

2 Type size

Even with a precise definition of the typographic point, there remains a problem: the definition of type size is not standardized; various *de facto* standards exist. The problem is to define what is to be measured. With hot metal, the obvious

way was to measure the body size which is common to all characters – capitals or lower cases – belonging to a given font. This is still the most used system. However it does not work any more: some manufacturers add extra white spaces so that ascenders or accentuated capitals of one line don't touch the descenders of the previous line; alas the size of these blanks varies from one font foundry to font foundry [Tracy86].

Today's digital characters have ceased to have a body with a physical existence. New proposals for defining type size have been made, but the tradition of body size still remains very strong. This notion will be discussed in further detail at the end of the present chapter.

3 Metric files

Let us quickly go through the tasks of a page description language interpreter or display driver generating page images in raster memory (for more details, see [AndréHersch92]). Let us analyze for each of these tasks which metric-related information is required. Almost all printing engines use the following font processing algorithm:

- **character selection:** each character description is selected by its name and by its code
- **rendering:** a so-called outline and filling algorithm is called that decides which pixels of a bitmap have to be blackened (see [Hersch92]). Bitmap fonts do not need to go through this step[4].
- **caching:** The final character bitmap is saved into cache memory before being copied upon request into page image.
- **spacing:** The starting position of the next character is updated.

Character selection is mainly concerned with names (encoding tables) and does not require metric information. **Rendering** works with contour descriptions (like Bézier curves) which, apart from scaling factors, are completely self contained. **Caching** does not need any special metric file since the scan conversion algorithm may be able to compute the required size of the cache memory used to

[4]Bitmat fonts follow metric format such as *BDF–Bitmap Distribution Format*.

store the produced character bitmap. However, for reasons of computation speed, most font machineries compute cache memory size by asking for the size of the corresponding character bounding-box[5].

Finally, only **spacing** is concerned with metrics. The metrics required to update the current position to that of the origin of the next character pertain to the traditional values such as width, kerning correction or side bearings. These values were included in the width tables of second generation phototypesetters. Today, with raster printers or phototypesetters, these values are sparsed into the glyph definitions. They have to be made explicit in metric files because desktop publishing programs (Word5, Framemaker, etc.) require exact knowledge of metrics in order to perform justification and hyphenation.

Today, a computer font is made up of at least two parts[6]. A first file contains the description of the glyphs (including hinting properties). This file is used by the printing engine. The second file contains the metrics associated with the font. It is used by application programs executing formatting tasks, such as those described above.

At printing time, when the PostScript program runs on the printer, metric files are not available. However, access to most of these values is possible either through font dictionaries (such as the PostScript `FontInfo` dictionary) or through page description language operators[7]. These metric files use formats that have not yet been definitely standardized. Among the most important formats, let us quote TFM (TEX Font Metrics [Rokicki85]), Ikarus [Karow87], Type 1 [Adobe90] and TrueType [TrueType90]. Relevant standards are SPDL and [ISO9541]. All of them have almost the same features.

Let us analyze the Adobe AFM *(Adobe Font Metrics)* file. First, we will examine how this file is organized and what its content is. Figure 3 gives an example of an AFM file.

---

[5]Note that PostScript offers two procedures to call the font machinery: one, `setcachedevice`, requires the bounding box while the other one, `setcharwidth`, does not.

[6]Many systems need a third file for display bitmaps. Font servers are used to handle these files; e.g. the Macintosh FOND resource handles properties of all the fonts belonging to a given typeface.

[7]For example, the width of a character may be computed thanks to the operator `stringwidth` while its bounding-box may be computed by traversing its contour description, using the `flattenpath` operator.

4 AFM organization

A metric file is a set of information either global to a font, or specific to each character. The AFM file (figure 3) contains four parts. The first part contains information global to the font, either related to the metrics (e.g. height of the capital line, angle of italic characters, etc.) or of general interest (family name, copyright, etc.). The second part contains information relevant to each of the characters (bounding-box) and indication of existing ligatures. Information about kerning pairs and about track kerning (not present in the present example) resides in the third part. The last part of the AFM file contains information about composite characters.

5 Horizontal control

Font metrics are mainly used for horizontal control of character placement.

5.1 Individual character metric

Characters are generated at the right place on the page image in raster memory by placing their origin at the currrent position. The current point is then updated in order to become the origin of the next character to be printed. Figure 2 exhibits the main metric values[8] for painting and spacing a character.

x, y	coordinates of the origin of the character
x', y'	coordinates of the next character
w_x, w_y	character width[9]
ur_x, ur_y, ll_x, ll_y	coordinates of the *bounding box*

w_x, w_y enable the character starting positions to be updated while ur_x, ur_y, ll_x, ll_y are only used for caching the bitmap[10]. These six values are the only ones to enter the metric file.

[8]These are the values commonly used by all authors such as [Black90], [Karow87], [Rockicki85], etc. However, some authors used different definitions.

[9]Here $w_y = 0$ for latin characters. For Hebrew or Arabic, $w_x < 0, w_y = 0$; for Chinese $w_x = 0, w_y < 0$, although the same font may have two sets of metrics in order to allow both horizontal and vertical justification modes [Adobe91], p.272.

[10]The caching algorithm needs not only each character box, but also the union of all these boxes that appears, as `FontBBox` in the AFM (see figure 3, part I).

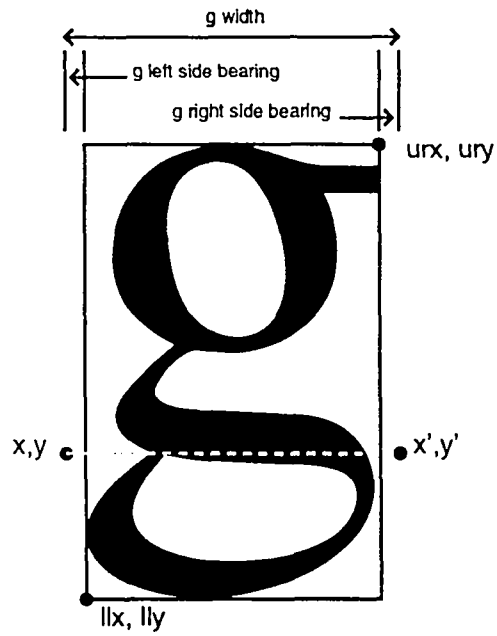

Figure 2: Individual character metrics as seen by PostScript. x, y are the coordinates of the origin of the character, x', y' those of the next character. Here $w_y = 0$. Note that PostScript ignores the body size.

In part II of figure 3 it can be seen that the character "g" (sixth line, with code 103) has $w_x = 500$ ($w_y = 0$) and its bounding-box has as coordinates $ll_x = 24, ll_y = -220, ur_x = 469, ur_y = 468$. All those values are given with a scaling factor of 1000 and can be modified at printing time by concatenating new affine transformations to the current transformation matrix.

Note that the side bearings do not belong to the metrics file. However, they may be computed as follows (with $w_y = 0$):

$$\begin{array}{ll} \text{left side bearing} & b_l = ll_x \\ \text{right side bearing} & b_r = w_x - ur_x \end{array}$$

The values of side bearings may be either positive or negative (or both, like for an italic "*f*").

5.2 Pair kerning

It is well known that the space between certain pairs of characters has to be shortened to avoid having an extra blank space between them ("AV" is better than "AV"). Metal types were mortised; PostScript characters are simply shifted, like in:

```
(A) show
-54 0 rmoveto
(V) show
```

This value of 54 could be decided by a formatter. However, type designers have a better feeling for the typeface than users and can therefore give appropriate values for the metric file. Figure 3 (part III line KPX A V) shows that this value 54 (when using a scaling factor of 1000) is the recommanded value when kerning A V with Mydidot italic[11]. High quality fonts have a very large set of such pair kerning values (not only for "AV" but for "r.", "r-", "LT", etc.).

5.3 Composite characters

Some European[12] characters can be considered to be made up of different parts. For example, a Ecircumflex "Ê" can be drawn with a "E" and, over it, a circumflex accent. Such a division saves space in font dictionaries. To allow that, the metrics file (e.g. fig. 3, part IV) contains the name of the new character (Ecircumflex), the first character to be used (E with a 0 0 offset), the second one to be added (circumflex with a translation $x = 0, y = 186$), etc. (a character may be composed of more than two others).

[11]Note that this value is not available in any font dictionary when runing a PostScript program. It is present only in the AFM, i.e. outside of the real PostScript world. See below 5.5.

[12]Non Indo-European characters (such as Kanji ones) may be considered as hierarchical composite characters. This subject exceeds the scope of this note. For more details, refer to [Adobe91], section 5.9.

5.4 Line justification

The width and pair kerning values allow any formatter to provide a rather good horizontal justification. Nevertheless, an additional technique, considered as heretic by traditional typographers, consists of varying the blanks between each pair of characters. This technique is called *track kerning*. It is related to the `awidthshow` operator: if the the font creator agrees with this technique, a given value, c_x (supposing $c_y = o$ for latin letters) appears in the font metrics file. For example, the *Times-Roman* AFM contains the following line:

```
TrackKern -3 6 -.1 72 -3.78
```

which means: if you want tight kerning (value -3), up to 6 points, decrease the space between two letters by $c_x = .1$ point; between 6 and 72 points, decrease by a linear factor comprised between .1 and 3.78; after 72 points, use $c_x = 3.78$ points.

5.5 Spacing on the fly

Spacing with width tables is the old technique used with hot metal where width values were only average values: indeed, it was not possible to punch all the types with all the combinations of side bearings depending on the next character, e.g. to punch a "a" with the perfect space when followed by a "a", a second "a" with the perfect space when followed by a "b", etc. Today computers allow better spacing by considering that each pair of characters mn may be separated by a space $S_{m,n}$ depending on m and on n. This method requires large tables, which however may be compressed (especially if topological properties are used). For this purpose, PostScript offers an operator: `kshow`. Let us consider the following elements:

T a table (actually a dictionary of dictionaries) such that $T_{i,j}$ is the correct bearing value between characters c_i and c_j (c_i, c_j are the coded integer values of those characters)

S is a procedure that finds the character codes c_j and c_i on top of the stack, removes these values and upgrades the current point by $T_{i,j}$

a font in which each character width is replaced by its bounding-box abscissa ($ur_x - ll_x$).

Then the following instruction

S ($c_0 c_1 c_2 \ldots c_n$) kshow

works as follows:

1. c_0 is placed (show) at the current point and the current point abscissa is updated by the c_0 character bounding-box abscissa.
2. The codes c_0 and c_1 are pushed on the stack and S is called: S removes these values and increments the current point abscissa by the value $T_{0,1}$.
3. kshow continues, shows c_1, updates the current point with the c_1 bounding-box abscissa, pushes c_1, c_2 on the stacks, calls S which upgrades the current point by $T_{1,2}$. .....
4. Finally, c_n is shown and the current point is updated with the c_n bounding-box abscissa.

This $T_{i,j}$ table may even be a $T_{i,j,b}$ table where each $T_{i,j}$ value depends on the current bodysize (or scaling factor): this allows real spacing on the fly.

Some commercial products work with such a system [URW92]. Alas, there is no standard to handle width pairs in today's metrics files. Futhermore, ongoing research aims at automatically computing the visual space between any pair of characters [Betrisey92].

```
StartFontMetrics 2.0  ************ part I ************
Comment Copyright(c) 1992 Didot project
Comment Creation Date:Tue July 14, 1992
FontName Mydidot-Italic
EncodingScheme AdobeStandardEncoding
FullName Mydidot Italic
FamilyName Mydidot
Weight Medium
ItalicAngle -13
IsFixedPitch false
UnderlinePosition -28
UnderlineThickness 62
Version 001.000
Notice Copyright(c) 1992. %Generaly here is information like
%Mydidot is a registered trademark of Myfoundry corp.
FontBBox -256 -220 1040 907
CapHeight 721
XHeight 448
Descender -212
Ascender 754
StartCharMetrics 244  ************ part II ************
C 32; WX 326; N space; B 0 0 0 0;
C 33; WX 158; N exclam; B 97 -13 256 703;
...
C 101; WX 482; N e; B 48 -12 486 498;
C 102; WX 412; N f; B 21 -215 508 738; L f ff; L i fi; L l fl;
C 103; WX 500; N g; B 24 -220 469 468;
...
C -1; WX 482; N zcaron; B 48 -12 486 498;
EndCharMetrics
StartKernData  ************ part III ************
StartKernPairs 115

KPX A W -54
KPX A V -54
KPX A T -68
...
KPX y comma -88
EndKernPairs
EndKernData
StartComposites 58  ************ part IV ************
...
CC Ecircumflex 2; PCC E 0 0; PCC circumflex 0 186;
CC Edieresis 2; PCC E 0 0; PCC dieresis 0 186;
...
CC aring 2; PCC a 0 0; PCC ring 24 0 ;
EndComposites
EndFontMetrics
```

Figure 3: An example of an AFM file (truncated) for an imaginary font (Mydidot)

6 Vertical control

PostScript, like other Page Description Languages, controls only the horizontal (for latin types) setting of characters. Vertical control (e.g. control of the distance between lines) is not provided. It is the formatter's task to choose the best interline space according to the current body sizes.

However, under certain circumstances such as putting characters into boxes, tight line spacing, underlining, etc. a formatter should know some vertical properties of the characters. Such properties are partly described in the AFM. They are related to the font as a whole and therefore located in the first AFM file part (see figure 3).

6.1 Horizontal lines

For a given font, all ascenders have the same height and all descenders the same depth. The AFM contains the position of the reference lines relative to the baseline (see figure 4).

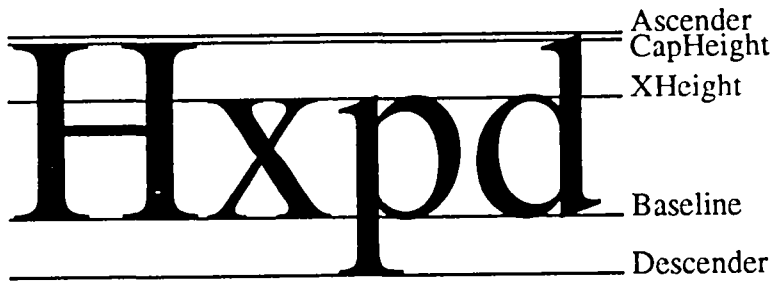

Figure 4: Horizontal lines are characteristic of a given font

Fine typography needs more lines than the basic reference lines. Capital, x-height, and base overhang lines are required for optical correction. These alignment lines, known in Type 1 as *blue lines*, are used only for character rendering and are therefore not part of the AFM. They are defined in a private dictionary (see [Gonczarowski92]).

6.2 Underlining

Again, the type designer should decide where characters can be underlined. AFM proposes the values `UnderlinePosition` and `UnderlineThickness` for that purpose. These are global values of a given font.

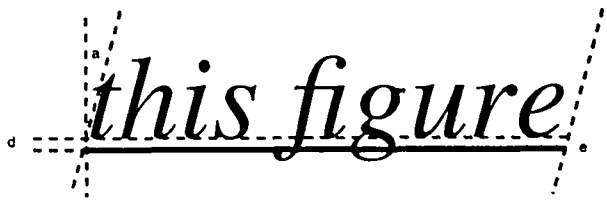

Figure 5: Italic angle (a), suggested underline position (d) and thickness (e) are common to all characters of a given font.

7 Other values

7.1 Ligatures

A ligature is a character that can be used to replace a sequence of two (or more) characters, in general for historical (calligraphic tradition) reasons. The best known examples are "fi" or "ffi" that respectively replace "fi" or "ffi".

However these ligatures have never entered the standard encoding tables (such as ISO8859 known as ISO–latin). Any font may have its own set of ligatures. The formatter must know which ligatures exist in order to automatically replace "fi" by "fi"[13]. Such information is given by adding the set of available ligatures for a given font to the metric file.

Ligatures are specified in the AFM by indicating, after the width values of the first character, which combination of other characters may be suitable for a ligature. For example, in figure 3 part II, line C 102, it can be seen that "f" may be ligatured (code L) with an "f" to give the character whose name is `ff`, with a "i" (character `fi`) or with a "l" (character `fl`).

[13] Actualy only few formatters such as TEX are able to generate ligatures. Ligatures are called by using internal character codes.

7.2 Miscellaneous

Among other features, let us quote:

- The angle of italic characters (this is a value global to a given font), see figure 3 part I and figure 5. Other formats, such as TFM, allow, for each character, to indicate the so-called "italic correction", i.e. the space to add between this italic character and a roman character (compare "*f*)" and "*f*)").
- Some information on the boldness or weight of the characters, see figure 3 part I. Note that this information is not quantified and terms like "medium, extra-bold, etc." are used.
- Some properties to handle mathematical formulae (e.g. TFM – see [Cousquer91]).

8 Conclusion

Hot metal typesetting required putting together types whose metrics included both the shape of the character (the part to be printed) and the space around it. When dealing with characters, today's printers and typesetters only take into account the printable black area.

Therefore, traditional notions such as "body size" have no real existence anymore. Scaling factors are used instead. However, measurements must be defined for reference size 1. Point size is not only a "vertical" metric; it is also used for "horizontal" control (width) of character placement. In fact, it can be considered as a reference to a surface. Moreover, widths are computed in units relative to the *em square*, which is the area covered by a square whose side is equal to the bodysize (quite often it is the width of "M").

It appears today that the best measurable element is the height of the capitals, sometimes called the *caps height* or *H-height.* A constant relationship between body size and H-height can be computed as shown in figure 6. And finally, there is a suggestion [Karow87] p.32, not to consider the body size in points but rather the cap height in millimeters.

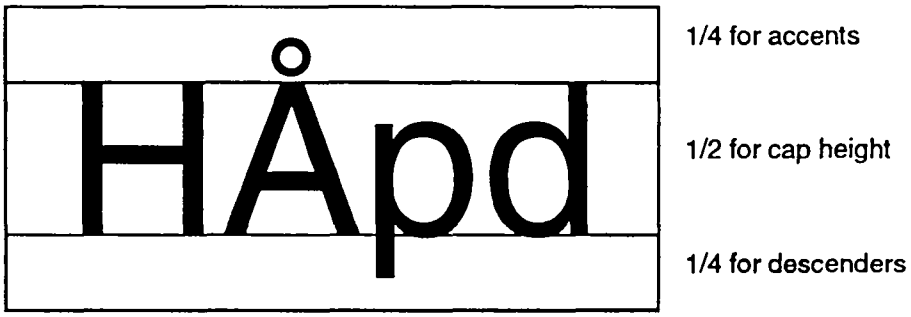

Figure 6: Karow's proposal for a new em square division

- the ratio of cap-height to bodysize is 2/3
- similarly, 1 point × 2/3 = 0.375 × 2/3 = 0.25 mm
- Thus, one obtains the cap height in mm by dividing the body size in typographic points by 4:

4 point bodysize = 1 mm cap height

Futhermore, a new em square division is proposed (see figure 6):

new body size = old body size × 1.33 = 2.0 cap height

Several companies [Holthusen90] propose these new measurement units. Results have been satisfactory so far.

References

[Adams89] Debra Adams and Richard Southall, "Problems of quality assessment", in André & Hersch (eds.), *Raster Imaging and Digital Typography*, Cambridge University Press, 1989, 213–222.

[Adobe90] Adobe Systems Incorporated, *Adobe Type 1 Font Format*, version 1.1, Addison-Wesley Publishing Company, Reading, MA (USA), 1990.

[Adobe91] Adobe Systems Incorporated, *PostScript Language Reference Manual*, Addison-Wesley Publishing Company, Reading, MA (USA), 2nd edition, 1991.

[André91] Jacques André & Justin Bur, Métrique de fontes PostScript, *Cahiers GUTenberg*, 8, March 1991, 29–50.

[AndréHersch92] Jacques André & Roger Hersch, "An introduction to digital type", in Roger Hersch (ed.), *Visual and Technical Aspects of Types*, Cambridge University Press, to appear, 1992.

[Betrisey92] Claude Betrisey and Catherine André, "An Enhanced PostScript Previewer for experimenting and teaching new approaches in digital typography", *Teaching Electronic Publishing. Bigre 79*, 1992, 83–87. To appear in *Electronic Publishing*.

[Black90] Alison Black, *Typefaces for desktop publishing – a user guide*, Architecture Design and Technology Press, London, 1990.

[Cousquer91] Alain Cousquer & Éric Picheral, "Polices, TEX et Cie", *Cahiers GUTenberg*, n° 9, juillet 1992, 3–31.

[Gonczarowski92] Jakob Gonczarowski, "Industry Standard Outline Font Formats", in Roger Hersch (ed.), *Visual and Technical Aspects of Types*, Cambridge University Press, to appear, 1992.

[Hersch92] Roger Hersch, "Font Rasterization", in Roger Hersch (ed.), *Visual and Technical Aspects of Types*, Cambridge University Press, to appear, 1992.

[Holthusen90] Bernd Holthusen and Albert Jan Pol, *Scangraphic Digital Type Collection*, Edition 4/1990, Mannesmann Scangraphic, tome 1.

[Karow87] Peter Karow, *Digital Formats for Typefaces*, URW Verlag, Hamburg (RFA), 1987.

[ISO9541] ISO, International Organization for Standardization, *Information processing – Font and character information interchange – Part 5: Font attributes and character model*, draft ISO/IEC/DIS 9541-5, 1987.

[Rokicki85] Tomas Rokicki, "Packed (PK) font file format", *TUGboat*, vol.6 (3), 1985, 115–130.

[Rubinstein88] Richard Rubinstein, *Digital Typography, An Introduction to Type and Composition for Computer System Design*, Addison-Wesley, Reading (USA), 1988.

[Tracy86] Walter Tracy, *Letters of credit: a view of type design*, Gordon Fraser, Londres 1986.

[TrueType90] Apple Computer Inc., *The TrueType Font Format Specification*, version 1.0, APDA M0825LL/A, 1990.

[URW92] URW Company, "The hz-Program," *Ikarus aktuell*, I/92,

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 666 AGREGATION FAIBLE DES PROCESSUS DE MARKOV ABSORBANTS
James LEDOUX, Gerardo RUBINO, Bruno SERICOLA
Juillet 1992, 30 pages.

PI 667 MODELES D'EVALUATION DE LA FIABILITE DU LOGICIEL ET TECHNIQUES DE VALIDATION DE SYSTEMES DE PREDICTION : ETUDE BIBLIOGRAPHIQUE
James LEDOUX
Juillet 1992, 76 pages.

PI 668 TWO COMPLEMENTARY NOTES ON SKEWED-ASSOCIATIVE CACHES
André SEZNEC
Juillet 1992, 10 pages.

PI 669 PARALLELISATION D'UN ALGORITHME DE DETECTION DE MOUVEMENT SUR UNE ARCHITECTURE MIMD
Fabrice HEITZ, Sergui JUFRESA, Etienne MEMIN, Thierry PRIOL
Juillet 1992, 34 pages.

PI 670 UN RESEAU SYSTOLIQUE INTEGRE POUR LA CORRECTION DE FAUTES DE FRAPPE
Dominique LAVENIER
Juillet 1992, 120 pages.

PI 671 EARLY WARNING OF SLIGHT CHANGES IN SYSTEMS AND PLANTS WITH APPLICATION TO CONDITION BASED MAINTENANCE
Qinghua ZHANG, Michèle BASSEVILLE, Albert BENVENISTE
Juillet 1992, 32 pages.

PI 672 ORDRES REPRESENTABLES PAR DES TRANSLATIONS DE SEGMENTS DANS LE PLAN
Vincent BOUCHITTE, Roland JEGOU, JeanXavier RAMPON
Juillet 1992, 8 pages.

PI 673 AN EXCEPTION HANDLING MECHANISM FOR PARALLEL OBJECT-ORIENTED PROGRAMMING
Valérie ISSARNY
Août 1992, 36 pages.

PI 674 A CALCULUS OF GAMMA PROGRAMS
Chris HANKIN, Daniel LE METAYER, David SANDS
Juillet 1992, 32 pages.

PI 675 EVALUATION DES PERFORMANCES D'UN NOYAU DE SIMULATION REPARTIE
Philippe INGELS, Carlos MAZIERO
Septembre 1992, 36 pages.

PI 676 FONT METRICS
Jacques ANDRE
Septembre 1992, 20 pages.

ISSN 0249-6399